Basics of Image Processing and ML in Python

TRACK MODULE I

Author: Ekaterina Golubeva

Supervisor: Prof. Dr. Thomas Ott

Institution: ZHAW

Date: Autumn semester 2022

Sources:

https://www.youtube.com/@DigitalSreeni/videos

https://github.com/bnsreenu/python_for_microscopists

Contents

1. Introduction: reading images using different Python libraries

1.1 WHY DO YOU NEED TO LEARN PROGRAMMING?

Programming is important for Life sciences because data is everywhere and under different forms (images, data, text). Programming helps extract information from this data. Software like ImageJ can help us to do the image processing. However, in order to treat big amount of images or treat them many times, it's much easier to create a code that does it and automate the task for us in stead of doing it manually. We collect repeatable information from our images so we get statistically similar results when we automate a task.

1.2 WHAT IS A DIGITAL IMAGE?
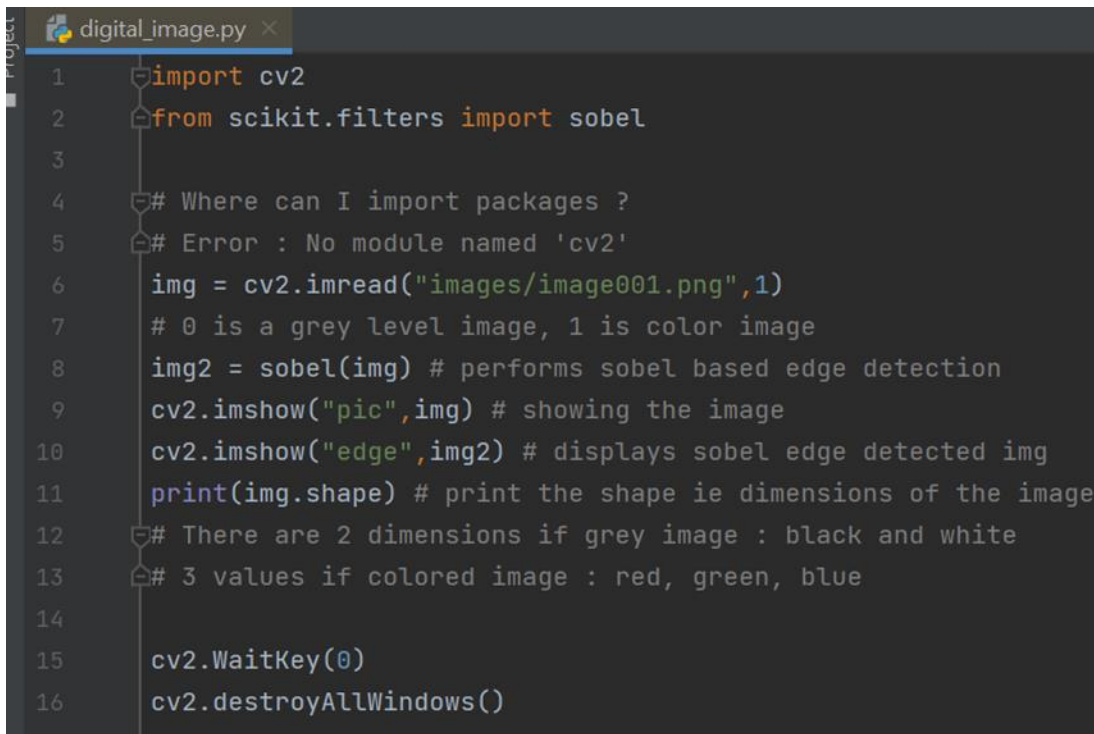
It's a collection of numbers in a matrix form, each number means an intensity, the intensity of that specific pixel, or coordinates in the color space RGB.

Example of a software : ImageJ allows to process for example and if you would like to sharp and smooth this image it's just one click so it's a great piece of software.

So a digital image is nothing but a bunch of numbers by manipulating the numbers we can actually change certain properties of the image. We can extract certain information, adjust brightness and contrast etc.

We can blur or sharp the image, some math is applied to this image. Every time we do some of these operations it's actually applying some sort of a mathematical algorithm on top of this image for blurring.

I may have for blur it's actually Applying some sort of a matrix at every pixel matrix multiplication at every pixel. Luckily we don't have to understand all of that, with only one click in this software it's pretty easy to do these things but how difficult is it with Python ?

```
digital_image.py ×
1    import cv2
2    from scikit.filters import sobel
3
4    # Where can I import packages ?
5    # Error : No module named 'cv2'
6    img = cv2.imread("images/image001.png",1)
7    # 0 is a grey level image, 1 is color image
8    img2 = sobel(img) # performs sobel based edge detection
9    cv2.imshow("pic",img) # showing the image
10   cv2.imshow("edge",img2) # displays sobel edge detected img
11   print(img.shape) # print the shape ie dimensions of the image
12   # There are 2 dimensions if grey image : black and white
13   # 3 values if colored image : red, green, blue
14
15   cv2.WaitKey(0)
16   cv2.destroyAllWindows()
```

First line: I'm just importing a library that I want to process the images with.

The second line: I'm going to read an image which is located in a folder in my computer.

I'm reading this image as a gray level image (=0) and if I change this to value to 1 it would be reading it as color. The next line I'm showing an image I'm showing displaying the image right on the screen and then I'm printing the shape of this image and the last two lines are there so I can kill this display window so when I run this code the image is open again this is opened as a gray level image.

1.3 UNDERSTANDING DIGITAL IMAGES FOR PYTHON PROCESSING

We've seen that an image is a number array. For each pixel we have values for red/green/blue plus the value alpha that defines the transparency of the picture. We usually work in RGB space (red/green/blue) but there's also Hue/Saturation/ Luminance space and we can transfer from one to the other in Python.

We can convert images type from one to another depending on our needs. Be careful, if we transform from uint16 to uint8 we'll lose information.

Red [255,0,0] Green [0,255,0] Blue [0,0,255] Yellow [255, 255,0] White [255,255,255]

Black [0,0,0]

```python
from skimage import io # work with images
import numpy as np # work with arrays
from matplotlib import pyplot as plt # plot images
from skimage import img_as_float # convert from one type to the other
# above are the conventional calls for most used libraries

my_image = io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")
print(my_image) # should print numpy array
print(my_image.min(), my_image.max()) # 0 and 255
plt.imshow(my_image)
plt.show()
print(type(my_image))
# Type : uint8 unsigned integers 8 bits
# Values go from 0 to 255 for each colour
# The (0,0) pixel is the top left pixer.
# It will be the first value we get from the print
```

```python
my_float_image = img_as_float(my_image) # changing type
print(my_float_image.min(), my_float_image.max()) # 0 and 1
plt.imshow(my_float_image) # they look exactly the same !
plt.show()
# Let's create an artificial image and fill in the value with some random numbers

random_image = np.random.random([500,500])
plt.imshow(random_image)
plt.show()
print(random_image)

# This is floating point 64 type image,
# so values are between 0 and 1
print(random_image.min(), random_image.max()) # 0 and 1

small_image = my_image*0.5 # reducing all values by half
print(small_image)
print(small_image.max())
plt.show()
```

1.4 READING IMAGES IN PYTHON

In this video, we have an introduction to different libraries to read and handle data. There are primarily four libraries in Python that can actually handle image processing : pillow pil, mat plot lib ,scikit image and OpenCV

```python
# pillow
# pip install pillow
from PIL import Image
import numpy as np

img = Image.open("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")
print(type(img)) # jpeg
# In pil image is not a numpy array !
# You have to transform it to numpy if you need numpy.

img.show()
print(img.format) # tells format # jpg
# Let's convert this into np type
img1 = np.asarray(img)
print(type(img1)) # numpy array
```

```python
# Matplotlib : plotting library
# Pyplot : generates plots

# pip install matplotlib
import matplotlib.image as mpimg # for short
import matplotlib.pyplot as plt

img = mpimg.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")
print(type(img)) # numpy array
print(img.shape) #
plt.imshow(img) # shows the image
plt.colorbar() # shows a colorbar next to the image
plt.show()
```

```python
import matplotlib.pyplot as plt
from skimage import io, img_as_float, img_as_ubyte
image = io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")

print(type(image)) # numpy array
plt.imshow(image)
# redefine image with float type, it converts the values
# from integers from 0 to 255 to normalized numers
# from 0 to 1 by dividing the values by 255
image_float = img_as_float(image)
print(image_float)
```

Reading several images, automate reading folder with images and change color space:

```python
# pip install glob
# to visualize several images

# Automate image opening and processing
import cv2
import glob
path = "C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\*"
# * stands for all the files
for file in glob.glob(path):  # reads all the files in the folder path
    print(file)
    # Original color RGB
    a = cv2.imread(file)
    print(a)
    cv2.imshow("Original Image", a)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    # BGR colors
    c = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
    cv2.imshow("Color Image", c)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

1.5 IMAGE PROCESSING USING PILLOW IN PYTHON

Pillow library for basic image processing: reading, rescaling, cropping, rotating, transforming …

Resize distorts the image to fit it to whatever size we tell it to resize. Thumbnail resizes but keeps the aspect ratio.

```python
from PIL import Image
# enables us to load images

# Open image
img = Image.open("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")
# opens the image. It's not a numpy array by default
print(type(img)) # JPEG
print(img.mode) #RGB
print(img.size) # weight and height in pixels

# Resizing
small_img = img.resize((200,300))
small_img.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\test_image_small.jpeg")
# It compresses the image, but it does't keep the aspect ratio
print(small_img.size)
img.thumbnail((1200,1300)) #resizes the original image
img.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\test_thumbnail.jpeg")
print(img.size) # it can't make the image bigger than the original size
```

```python
from PIL import Image

# Cropping
img = Image.open("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg")
cropped_img = img.crop((0,0, 300, 300)) #
cropped_img.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\cropped_img.jpeg")
```

```python
from PIL import Image

# Rotate an image
img = Image.open("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\image003.jpg")
img45 = img.rotate(45, expand = True) # rotate by 45 degrees
img45.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\rotated.jpeg")
# When we crop by 45 degrees we lose the edges, if we want to keep them,
# we have to add expand = True so the rotated image fits within the new window size

# Image flipping from left to right/ from top to bottom
imgh = Image.open("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\image001.jpg")
img_flipLR = imgh.transpose(Image.Transpose.FLIP_LEFT_RIGHT)
img_flipLR.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\flipLR.jpg")
img_flipTB = imgh.transpose(Image.Transpose.FLIP_TOP_BOTTOM)
img_flipTB.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\flipTB.jpg")


grey_img = imgh.convert("L") # convert to grey level
grey_img.save("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\grey.jpg")
```

Automate rotation of images in a folder with pillow

```python
1   from PIL import Image
2   import glob
3
4   # Automate rotation of images in a folder with pillow
5   path = "C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\*"
6   for file in glob.glob(path):
7       print(file)
8       a = Image.open(file)
9       rotated45 = a.rotate(45, expand=True)
10      rotated45.save(file+"_rotated45.png", "PNG") # save each file, not overwrite
```

1.6 IMAGE PROCESSING USING SCIPY IN PYTHON

It's a library which is part of the numpy stock. It contains modules for linear algebra, signal processing, image processing. It's not designed specifically for image processing. Scikit image is a library that contains a lot of operators.

```python
1   # from scipy import misc
2   from skimage import io, img_as_ubyte
3   from scipy import ndimage
4   import numpy as np
5   from matplotlib import pyplot as plt
6
7   img = img_as_ubyte(io.imread("images\\image002.jpeg", as_gray = True))
8   print(img.shape) # RGB image or gray image if as_gray = True
9   #print(img[10:15,20:25]) #subset of the image
10
11  mean_grey = img.mean()
12  max_value =img.max()
13  min_value = img.min()
14
15  print("Min, Max , and Mean are", min_value, max_value, mean_grey)
16
17  # Basic geometric operations
18  flippedLR = np.fliplr(img)
19  flippedUD = np.flipud(img)
```

```
21        # Subplots allows us to plot multiple plots on the same pannel
22        # cmaps are colors maps in matplotlib
23        plt.subplot(2,1,1) # columns, rows, images
24        plt.imshow(img, cmap= "Greys")
25        plt.subplot(2,2,3)
26        plt.imshow(flippedLR, cmap= "Blues")
27        plt.subplot(2,2,4)
28        plt.imshow(flippedUD, cmap= "hsv")
29        plt.show() # to display
```

```python
from skimage import io, img_as_ubyte
from scipy import ndimage
import numpy as np
from matplotlib import pyplot as plt
# Rotation
img = img_as_ubyte(io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\image003.jpg", as_gray = True))
rotated = ndimage.rotate(img, 90, reshape = False)
# reshape is True by default to keep the correct aspect ratio
plt.imshow(rotated)
plt.show()
```
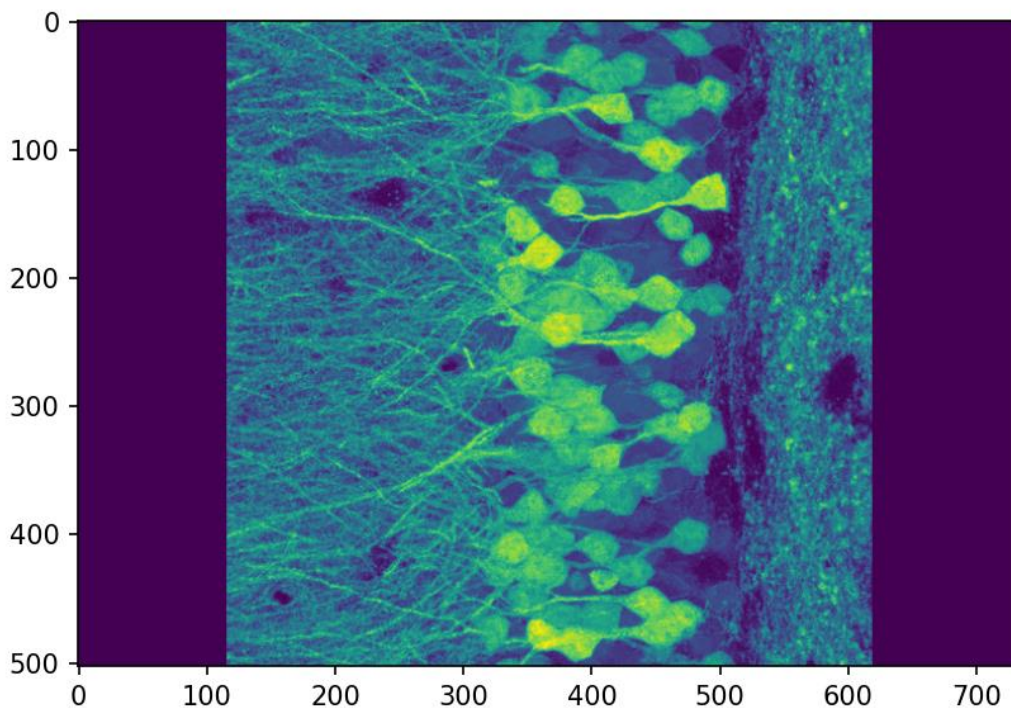

Figure 1 Rotated image

Filters: Gaussian, uniform, median, Sobel

```python
from skimage import io, img_as_ubyte
from scipy import ndimage
import numpy as np
from matplotlib import pyplot as plt
# Filters
img = img_as_ubyte(io.imread("images\\image002.jpeg", as_gray = True))
# Blurring filter
uniform_filtered = ndimage.uniform_filter(img, size =9)
plt.imshow(uniform_filtered)
plt.show()

# Gaussian smoothing/Blurring : it averages values, doesn't preserve the edges
gaussian_filtered = ndimage.gaussian_filter(img, sigma =7)
# sigma = number of Gaussian filters applied
plt.imshow(gaussian_filtered)
plt.show()

median_filtered = ndimage.median_filter(img, size =3)
plt.imshow(median_filtered)
plt.show()
```

```python
median_filtered = ndimage.median_filter(img, size =3)
plt.imshow(median_filtered)
plt.show()

sobel_img = ndimage.sobel(img, axis =0) # default axis = -1
plt.imshow(sobel_img)
plt.show()
```

1.7 Introduction to image processing using scikit-image in Python

Rescale, resize, downscale images.

```
1   from skimage import io
2   from matplotlib import pyplot as plt
3
4   img = io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\image001.jpg", as_gray_=True)
5   print(img.shape) #(450, 665)
6   from skimage.transform import rescale, resize, downscale_local_mean
7
8   rescaled_img = rescale(img,1.0/4.0,anti_aliasing_=_True)
9   resized_img = resize(img, (200,200),anti_aliasing_=_True_)
10  downscaled_img = downscale_local_mean(img, (4,3))
11  print(downscaled_img.shape)
12  #plt.imshow(rescaled_img)
13  #plt.imshow(resized_img)
14  plt.imshow(downscaled_img)_# (113, 222)
15  plt.show()
```

1.8 READING IMAGES IN OPENCV

OpenCV Computer vision type of applications facial recognition, motion tracking, microscopy images application, also for time series. We'll focus on following 5 topics in image processing

Splitting and merging channels

```
1   import cv2
2   from matplotlib import pyplot as plt
3
4   img =cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\brainbow\\image003.jpg",1)
5   print(img.shape) #image are in BGR by convention in opencv
6
7   blue, green, red = cv2.split(img) #short way of splitting image in 3 images
8   # where only bgr pixels are highlighted
9
10  img_merged =cv2.merge((blue,green,red)) # does the opposite of split
11  cv2.imshow("Merged",img_merged)
12
13  blue = img[:,:,0] # blue pixels
14  green = img[:,:,1] #green pixels
15  red =img[:,:,2] # red pixels
16  cv2.imshow("Blue pixels",blue)
17  cv2.imshow("Red pixels",red)
18  cv2.imshow("Green pixels",green)
19  cv2.waitKey(0)
20  cv2.destroyAllWindows()
```

```
1   import cv2
2
3   img = cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\test_image_small.jpeg")
4   resized = cv2.resize(img,None,fx=2, fy=2, interpolation_=_cv2.INTER_CUBIC)
5   # factor
6   cv2.imshow("Original image",img)
7   cv2.imshow("Resized image",resized)
8
9   cv2.waitKey(0)
10  cv2.destroyAllWindows()
```

1.9 LIST OF COMMANDS USED

Command	Description	Library/Package
import cv2	Import OpenCV	OpenCV
cv2.imread	Read an image	OpenCV
sobel(img)	Performs sobel based edge detection	OpenCV
cv2.imshow("name",img)	Shows the image with a chosen name	OpenCV
print(img.shape)	Prints the shape ie dimensions of the image	OpenCV
cv2.WaitKey(0) cv2.destroyAllWindows()	Closes windows 0 -manually N – after N milliseconds	OpenCV
plt.imshow(cv2.cvtColor(img ,cv2.COLOR_BGR2RGB))	Changes the color space	OpenCV
io.imread(img path, as_gray = True/False)	Reads the file	Skimage, io
img_as_ubyte()	Changes the type of image to ubyte	Skimage
plt.show()/plt.imshow(img, cmap= "Greys")	Plots the figure above Cmap is a colormap	matplotlib.pyplot
plt.subplot(2,1,1)	Create a subplot in a big panel of multiple plots	Matplotlib, pyplot
mpimg.imread()		matplotlib.image
img_as_float()	Changes the type of image to float	Skimage, img_as_float
np.random.random([n,m])	Creates random image with dimensions nxm	numpy
np.flipud(img)	Flips upside down	numpy
Np.fliplr(img)	Flips from left to right	numpy
Image.open()	Reads the image	Pillow
np.asarray()	Saves the image in np arra type (not jpg)	Pillow
glob.glob(path)	Reads all the files in the folder path	glob
img.resize((n,m))	Resizes image (without preserving aspect ratio)	Pillow
img.thumbnail((n,m))	Resizes the image preserving the aspect ratio	Pillow
img.save()	Save the image at the indicated path	Pillow
img.crop()	Crops image, indicate coordinates	Pillow
img.rotate(degrees, expand = True)	Rotates the image by degrees and fits the image so the edges are preserved	Pillow
imgh.convert("L")	convert to grey level	Pillow
img.transpose(Image.Transpose.FLIP_LEFT_RIGHT)	Flips the image from left to right	Pillow
imgh.transpose(Image.Transpose.FLIP_TOP_BOTTOM)	Flips the image from top to bottom	Pillow
ndimage.uniform_filter(img, size =n)	Uniform filter	Scipy/skimage
ndimage.gaussian_filter(img, sigma =n)	Gaussian filter	Scipy/skimage
ndimage.median_filter(img, size =n)	Median filter	Scipy/skimage
ndimage.sobel(img, axis =0)	Sobel filter	Scipy/skimage

2. Introduction to basic filtering operations

What are image processing filters?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is one of the most basic and common image operations in image processing. You can filter an image to remove noise or to enhance features; the filtered image could be the desired result or just a preprocessing step.

2.1 Denoising microscope images

There will always be come noise in our images, how can we clean this up?

Digital filters are a convolution between a kernel and your image, multiplication of two arrays of different sizes. A kernel is a small matrix (a linear filter like Gaussian, or nonlinear filters that preserve edges) and your image is much bigger one. Let's discuss 3 different denoising filters: Gaussian, median and nonlocal means filters.

Gaussian and Median:

```python
from matplotlib import pyplot as plt
import numpy as np
from skimage import io
from scipy import ndimage as nd
from skimage.restoration import denoise_nl_means, estimate_sigma

def gaussian_kernel(size,size_y=None):
    size=int(size)
    if not size_y:
        size_y=size
    else:
        size_y=int(size_y)
    x,y = numpy.mgrid[-size:size+1,-size_y:size_y+1]
    g = numpy.exp(-(x**2/float(size)+y**2/float(size_y)))
    return g/g.sum()
```

```python
gaussian_kernel_array=gaussian_kernel(1)
print(gaussian_kernel_array)
plt.imshow(gaussian_kernel_array,cmap=plt.get_cmap('jet'),interpolation
='nearest')
plt.colorbar()
plt.show()

# when you convolve an image with a kernel you don't change the
intensity/energy of the image
# we just multiply it by 1
# 2d Gaussian is a bell shape


img =
io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\ne
uron.jpg")
# Let's read a noisy image and apply gaussian filter
gaussian_img = nd.gaussian_filter(img,sigma =3)
plt.imshow(gaussian_img)
# The image is cleaner but blurred and we're loosing information
#plt.imsave("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\
gaussian.jpg", gaussian_img)
# Gaussian filter is typically not preferred, let's try median

median_img = nd.median_filter(img,size=3) # size of the kernel
plt.imshow(median_img)
plt.imsave("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\m
edian.jpg", median_img)
# much better denoising, edges are preserved
```

2.2 NON-LOCAL MEANS DENOISING METHOD

The method is based on a simple principle: replacing the color of a pixel with an average of the colors of similar pixels. But the most similar pixels to a given pixel have no reason to be close at all. It is therefore licit to scan a vast portion of the image in search of all the pixels that really resemble the pixel one wants to denoise

```python
from matplotlib import pyplot as plt
import numpy as np
from skimage import io, img_as_float
from scipy import ndimage as nd
from skimage.restoration import denoise_nl_means, estimate_sigma
```

```
img =
img_as_float(io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProjec
t\\images\\neuron.jpg"))
# Let's look at the non local means filter
sigma_est = np.mean(estimate_sigma(noisy, channel_axis=-1))
nlm = denoise_nl_means(img, h=1.15 * sigma_est, fast_mode = True,
                       patch_size=5,
                       patch_distance=6,channel_axis=-1)
plt.imshow(nlm)
plt.imsave("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\n
lm.jpg", nlm)
```

First, let's see the formula for NLM,

$$NL[v](i) = \sum_{j \in I} w(i,j)v(j),$$

The algorithm calculates the estimated value of a pixel as the weighted average of all the pixels in the image, but the family of weights depends on the similarity between the pixels *i* and *j*. In other words, it looks at a patch of image and then identifies other similar patches in the whole image and takes a weighted average of them. To understand this, consider the following image.
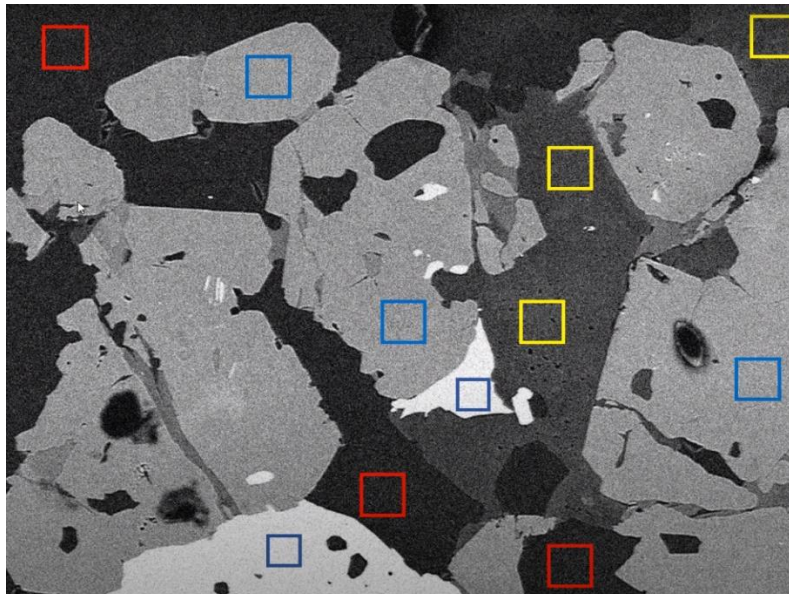
Here, similar patches are marked with the same-colored square boxes. So now, it will take the weighted average of pixels of similar patches as the estimated value of the target pixel. This algorithm takes as input the patch size and the patch distance. Consider the following grayscale image which has been denoised using an NLM filter.
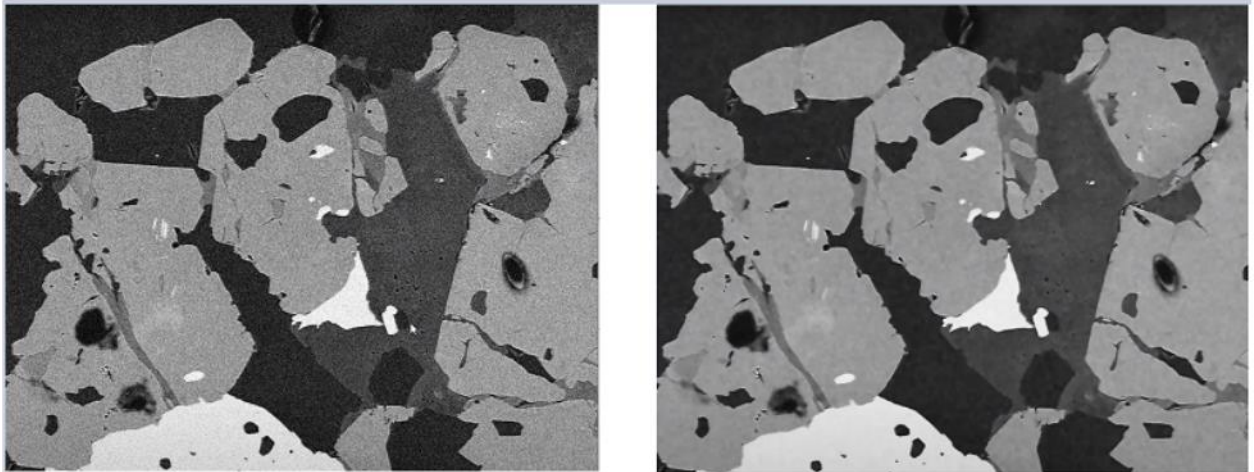
Left side—noisy image; Right side—denoised image using NLM filter

We can see that NLM does a decent job of denoising an image. One can notice a slight blur to the denoised image. This is because of the *mean*. A mean applied to any data will smooth out the values. However, when the level of noise is too high, NLM fails to provide good results.

2.2 HISTOGRAM BASED IMAGE SEGMENTATION

We take grey level images and segmentate regions that have different grey levels. Sometimes it doesn't work if we have same mean gray level but different textures – in this case we need to do some preprocessing like using entropy filter.

We'll firstly perform non-local means denoising as a first step and do a segmentation upon that.

Histogram equalization is used to improve the contrast of our images.

Consider an image whose pixel values are confined to some specific range of values only. For example, brighter image will have all pixels confined to high values. But a good image will have pixels from all regions of the image. So you need to stretch this histogram to either ends (as given in below image, from Wikipedia) and that is what Histogram Equalization does. This normally improves the contrast of the image.
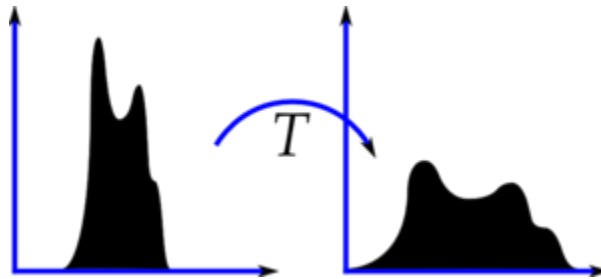
2.3 EDGE DETECTION OPERATORS

Edge operators are used in image processing within edge detection algorithms. They are discrete differentiation operators, computing an approximation of the gradient of the image intensity function.

Different operators compute different finite-difference approximations of the gradient. For example, the Scharr filter results in a less rotational variance than the Sobel filter that is in turn better than the Prewitt filter. The difference between the Prewitt and Sobel filters and the Scharr filter is illustrated below with an image that is the discretization of a rotation- invariant continuous function. The discrepancy between the Prewitt and Sobel filters, and the Scharr filter is stronger for regions of the image where the direction of the gradient is close to diagonal, and for regions with high spatial frequencies. For the example image the differences between the filter results are very small and the filter results are visually al-most indistinguishable.

Sobel operator

Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

The **Canny filter** is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

```python
import cv2
img =
cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\image002.jpeg",1)
edges = cv2.Canny(img,100,200)
cv2.imshow("Original",img)
cv2.imshow("Canny Filter",edges)
```
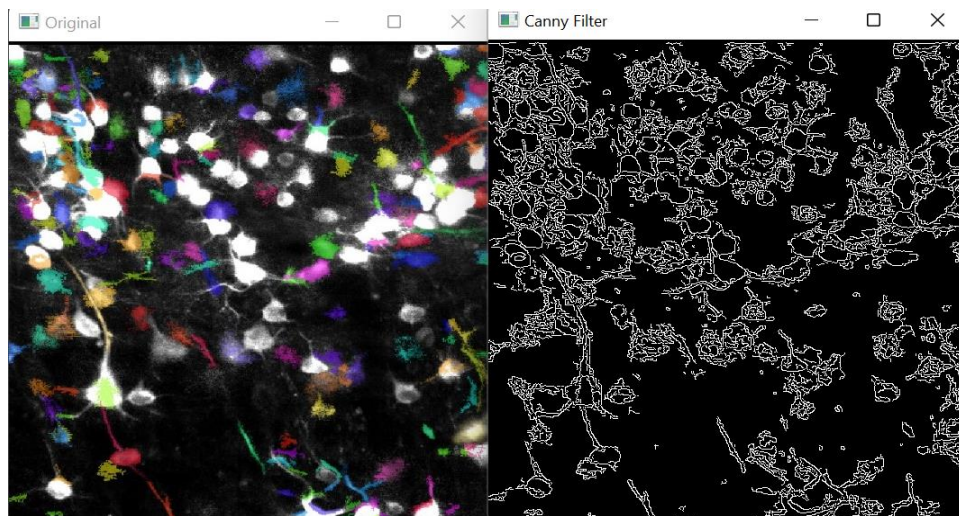
2.4 ENTROPY FILTER

In information theory, information entropy is the log-base-2 of the number of possible outcomes for a message. For an image, local entropy is related to the complexity contained in a given neighborhood, typically defined by a structuring element. In other words, entropy is the amount of disorder in your data The entropy filter can detect subtle variations in the local gray level distribution. Discrete entropy is used to measure the content of an image, where a higher value indicates an image with richer details. Shannon's entropy is defined as:

$$E\left(I\right) = -\sum_{k=0}^{L-1} p\left(k\right) log_2\left(p\left(k\right)\right)$$

where I is the original image, $p(k)$ is the probability of occurrence of the value k in the image I, and L=2q indicates the number of different gray levels. E(I) is a convenient notation for the entropy of an image, and should not be interpreted here as a mathematical expectation since I is not a random variable. It is not difficult to prove that if q is the number of bits representing each pixel in the image, then $E(I) \in [0,q]$; for this work q=8 for infrared thermal images in gray scale. In image processing, discrete entropy is a measure of the number of bits required to encode image data. The higher the value of the entropy, the more detailed the image will be. We would like to process the next image using the entropy filter:
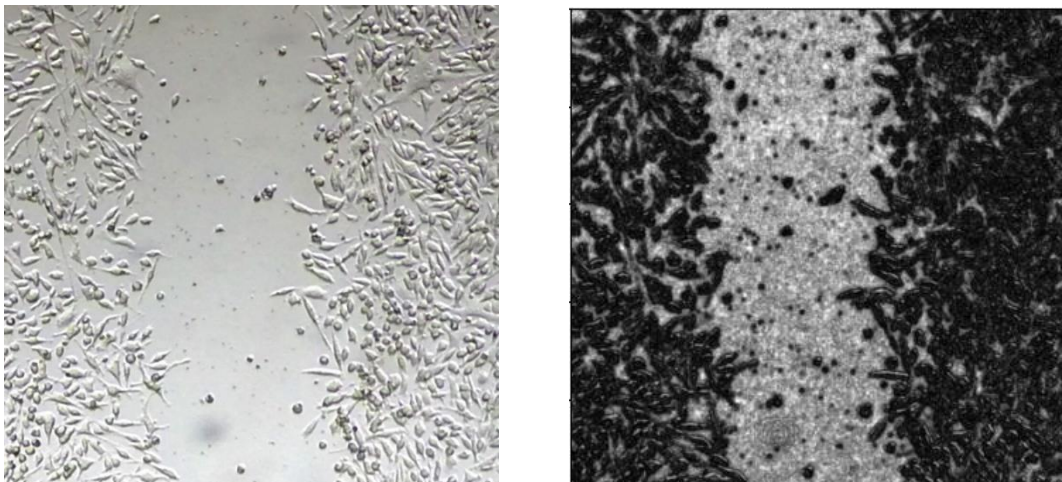

Figure 2 Left : Original image of healing wound. Right : Entropy filtered image

We would like to measure the area of the wound and see how it decreases with healing. Quantify the area of the clean image is not easy, it has the same level of grey as the surrounding area, we use texture filters and entropy is one of those. We decide that the disk - small area measure unit- is on size 3.

We can add thresholds segmenting these two regions. With entropy, by studying the texture of this picture, we can make it possible to add a threshold It gives us an image that is easy to segment. Let's do Otsu thresholding

Threshold doesn't give us an image it gives us a number, one value which corresponds to the threshold for this specific image. Let's then convert it into a binary image: binary image, a boolean type of image, is true if it's less then the threshold value, otherwise it is false. We then print out the percentage of white pixels.

```python
from matplotlib import pyplot as plt
from skimage import io
from skimage.filters.rank import entropy
from skimage.morphology import disk
from skimage import img_as_ubyte
from skimage.filters import try_all_threshold
from skimage.filters import threshold_otsu
import numpy as np
img =
img_as_ubyte(io.imread("C:\\Users\\golub\\PycharmProjects\\pythonProjec
t\\images\\healing_wound.jpg",as_gray=True))
entropy_img = entropy(img, disk(3))
#plt.imshow(entropy_img, cmap = 'Greys')
#fig, ax = try_all_threshold(entropy_img,figsize = (10,8), verbose =
False)
#plt.imshow()


thresh = threshold_otsu(entropy_img)
binary = entropy_img <= thresh
plt.imshow(binary, cmap ='gray')

print("The precent bright pixels is: ",
(np.sum(binary==1)*100/((np.sum(binary==1)+ np.sum(binary==0)))))
#The precent bright pixels is:  38.337643016542096
```

2.4 CLAHE AND THRESHOLDING USING OPENCV

Let's study enhancing images using histogram equalization HE vs CLAHE (Contrast Limited Adaptive Histogram Equalization) and thresholding.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
# Contrast Limited Adaptive Histogram Equilization (CLAHE) and thresholding

img =
cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\healing_wound.jpg",0)
eq_img = cv2.equalizeHist(img)

plt.hist(img.flat, bins=100,range=(0,255))
plt.hist(eq_img.flat, bins=100, range=(0,255)) #equalized histogram

clahe = cv2.createCLAHE(clipLimit =2.0, tileGridSize=(8,8))
cl_img = clahe.apply(img)
cv2.imshow("CLAHE image",cl_img) #less noise
cv2.imshow("Equalized image",eq_img)
cv2.imshow("Original image", img)
#considers global contrast of the image. It might not always work well.
# There's something called adaptive HE: it does HE in small patches
# it is contrast limited
cv2.waitKey(0)
cv2.destroyAllWindows()
```

After we have created a CLAHE image it is easier to threshold it.

Thresholding is used to create a binary image from a grayscale image. It is the simplest way to segment objects from a background.

Thresholding algorithms implemented in scikit-image can be separated in two categories:

•   Histogram-based. The histogram of the pixels' intensity is used and certain assumptions are made on the properties of this histogram (e.g. bi-modal).

•   Local. To process a pixel, only the neighboring pixels are used. These algorithms often require more computation time.

2.5 OTSU THRESHOLDING METHOD

Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold (the pixels that either fall in foreground or background). The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum. Otsu threshold will bias in favor of the class with larger variance when the within-class variances of two classes are substantially different.

Code : Threshold method s: Binary and OTSU
Binary : After seeing the histogram we can adapt the range to zoom in a region which can be segmented and decide the threshold value ourselves.
OTSU based threshold comes up with the threshold value automatically and it gives it as an outcome: here 129 pixel value. Arguments for thresh are image with CLAHE histogram, second is the threshold value identified with the histogram (for binary)

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\alloy.jpg",0)
plt.hist(img.flat, bins=100,range=(0,255))

clahe = cv2.createCLAHE(clipLimit =2.0, tileGridSize=(8,8))
cl_img = clahe.apply(img)

ret, thresh1= cv2.threshold(cl_img,129,200,cv2.THRESH_BINARY)
ret2, thresh2= cv2.threshold(cl_img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

cv2.imshow("Binary Threshold ",thresh1)
cv2.imshow("OTSU based thresholding ",thresh2)
cv2.imshow("Original image", img)
```
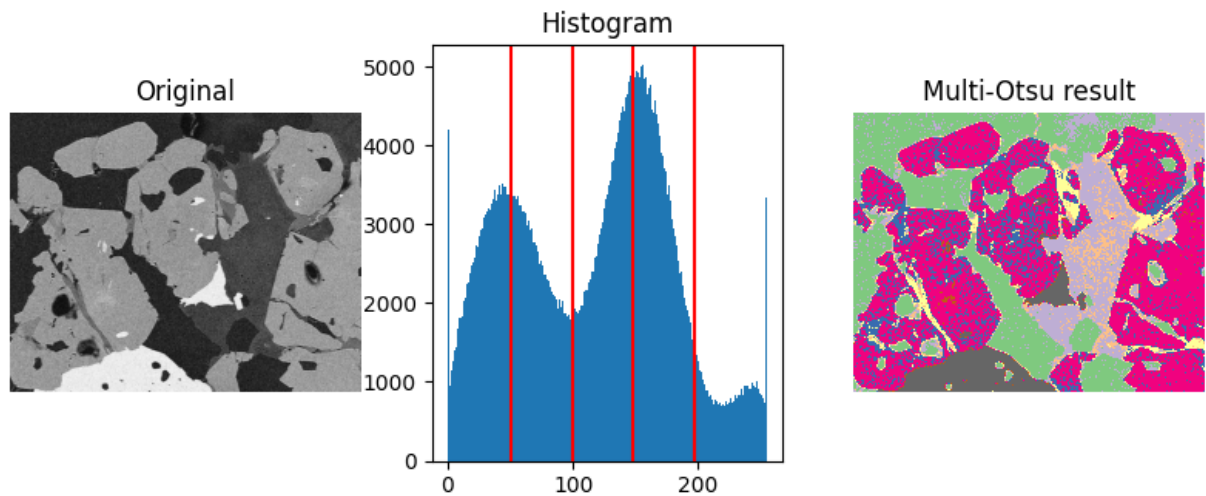
Remember to denoise first before thresholding and segmentation!

2.6 LIST OF COMMANDS USED

Command	Description	Library
cv.fastNlMeansDenoising(image)	Non local means denoising	OpenCV
otsu_threshold, image_result = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU,) threshold_otsu(entropy_img)	Otsu thresholding	OpenCV
cv2.GaussianBlur(image, (5, 5), 0)	Gaussian blur denoising	OpenCV
plt.hist(image.flat, bins=100,range=(0,255))	Histogram of an image	Matplotlib, pyplot
filters.roberts(image)	Roberts edge detecting filter	Scikit-image
filters.sobel(image)	Sobel edge detecting filter	Scikit-image
Filters.prewitt(image)	Prewitt edge detecting filter	Scikit-image
Command	Description	Library
entropy(img, disk(5))	Entropy	Scikit image, filters, rank
nd.gaussian_filter(img,sigma =3)	Gaussian denoising filter	Scipy, ndimage
nd.median_filter(img,size=3)	Median denoising filter	Scipy,

		ndimage
blue, green, red = cv2.split(img)	Split color channels	OpenCV
cv2.merge((blue,green,red))	Merge color channels	OpenCV
cv2.Canny(img,100,200)	Canny edge detector	OpenCV
cv2.equalizeHist(img)	Equilized histogram	OpenCV
clahe = cv2.createCLAHE(clipLimit =2.0, tileGridSize=(8,8)) cl_img = clahe.apply(img)	CLAHE	OpenCV
cv2.threshold(cl_img,129,200,cv2.THRESH_BINARY)	Binary thresholding	OpenCV

3. Introduction to Machine Learning

3.1 WHAT IS MACHINE LEARNING?
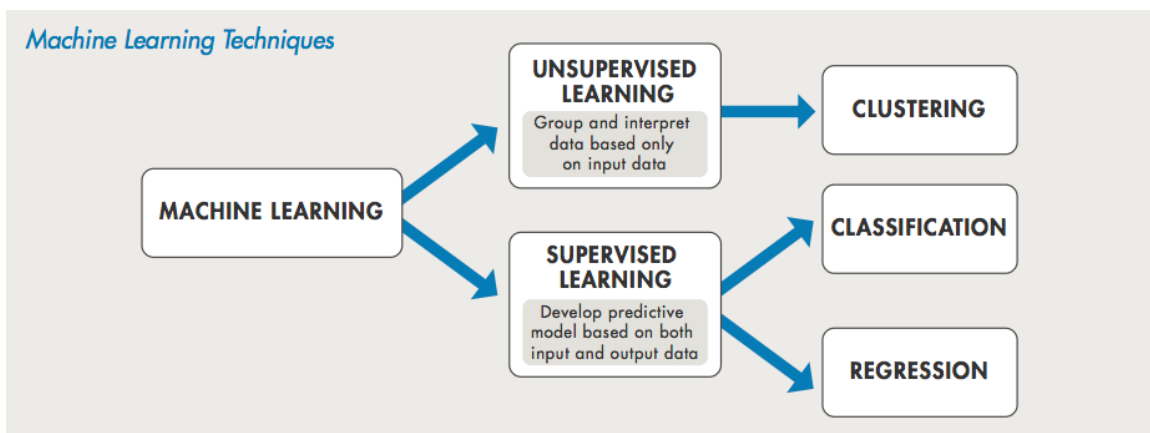
Humans use knowledge about objects to be able to identify them. We train models based on discoveries about the human brain to teach machines how to learn too. We want to automate processes with ML, AI, DL. We want to train it to automate tasks, for efficiency, productivity, gain of time and accuracy.

There are two types of problems : prediction and classification

1) Predict a continuous value with linear regression
2) Classify into discrete category (binary problem: cell or background for example)

Supervised learning: e.g. classification/segmentation. We know which pixels belong where, based on that information the machine learns. We can add feature, contrast texture information. We're providing the algorithm some information what we know about it, it learns and uses that info to predict future

unseen data.

Unsupervised learning : Clustering, we're not giving information. The algorithm clusters itself. We can give the number of clusters we want.

How to implement this? Here's the workflow:

- Split dataset for training and testing
  (training is for the algorithm to learn, we want to save some percent of the data for testing purposes)
- Train a model
- Make prediction on test dataset
- Compare predicted result with true values

The result might be in the range of 50%-90+% of accuracy. If It's accepted or not depends on the size of the dataset and the purpose.

Deep learning / Neural Networks

The main difference is in feature extraction in the training stage. One feature could be pixel value, the other could be the texture. We have to engineer the features in deep learning, this feature extraction is built in model training. When we have limited dataset classical ML has better results but with increasingly bigger once it's more efficient to use DL with already trained data.

3.2 SPLITTING DATA INTO TRAINING AND TESTING SETS

Let's say we have a scattered data. We assume a straight line could fit the data: y = ax +b

We decide for example that 60% of the data set is to train, 40% to test.

For linear regression, Y=the value we want to predict

X= all independent variables upon which Y depends.

There are 3 steps for linear regression....

        Step 1: Create the instance of the model

        Step 2: .fit() to train the model or fit a linear model

Step 3: .predict() to predict Y for given X values.

Now let us define our x and y values for the model.

x values will be time column, so we can define it by dropping cells

y will be cells column, dependent variable that we are trying to predict.

```
x_df = df.drop('cells', axis='columns')
#Or you can pick columns manually. Remember double brackets.
#Single bracket returns as series whereas double returns pandas
dataframe which is what the model expects.
#x_df=df[['time']]
print(x_df.dtypes)   #Prints as object when you drop cells or use double
brackets [[]]
#Prints as float64 if you do only single brackets, which is not the
right type for our model.
y_df = df.cells

X_train, X_test, y_train, y_test = train_test_split(x_df, y_df,
test_size=0.4, random_state=10)
#random_state can be any integer and it is used as a seed to randomly
split dataset.
#By doing this we work with same test dataset every time, if this is
important.
#random_state=None splits dataset randomly every time
```

3.3 WHAT ARE FEATURES IN MACHINE LEARNING?

Features are used in supervised ML, these are characteristics that describe our data.

We can decide that there are 5 different regions for example, we can use pixel value as feature, edges, local contrast etc. With applying different filters, we can get different results – features.

How do we select? Just looking at the image we can decide which features are the best for my image or generate 1000 different filtered images and let the machine learning algorithm figure out how to describe the different regions of interest (also using feature importance function).

Example:

Feature vector = size 11 : 1 with original pixel and 10 different filters for example.

A list of value at a given a pixel where each value is the value of the pixel with an applied filter.

The premise is that the combination of these filters will describe the region of interest.

For example, for a cell image each cell will have the same feature vector because it has the same structure.

How do we know which filters work? If we know which filters work, we don't need to use ML. If it's obvious (for example all cells are blue because of fluorescence) we can use classical techniques. But if there are different structures, ML can help to find the combination.

Which ML should you use? If you know what you're looking for – you're the feature engineer.

With limited dataset: traditional ML outperforms any other DP or NN. But if we have thousands of data which neurons already labelled, then DL is better.

How to put all this info together? Let's have a look at the implementation:

3.4 HOW TO GENERATE FEATURES IN PYTHON USING ML?

```
img =
cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\s
cratch.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

img2 = img.reshape(-1)
df = pd.DataFrame() # creating an empty data frame
df['Original Image'] = img2 # adding a new column in my data frame

entropy_img = entropy(img, disk(1))
entropy1 = entropy_img.reshape(-1) # unwrap into an 1D-array
df['Entropy'] = entropy1

gaussian_img = nd.gaussian_filter(img, sigma=3)
gaussian_img1 = gaussian_img.reshape(-1)
df['Gaussian s3'] = gaussian_img1
```

```
sobel_img = sobel(img)
sobel1 = sobel_img.reshape(-1)
df['Sobel'] = sobel1

print(df.head()) # gives individual feature vectors
cv2.imshow('Original Image', img)
cv2.imshow('gaussian', sobel_img)
cv2.imshow('entropy', entropy_img)
cv2.imshow('Sobel', sobel_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

For example, for the scratch image entropy is a good feature, others not. For cells entropy is not a good feature but Sobel, Gaussian are good filters. Have a look at the Figure 2. Try different filters and pick the ones that are good for your picture.
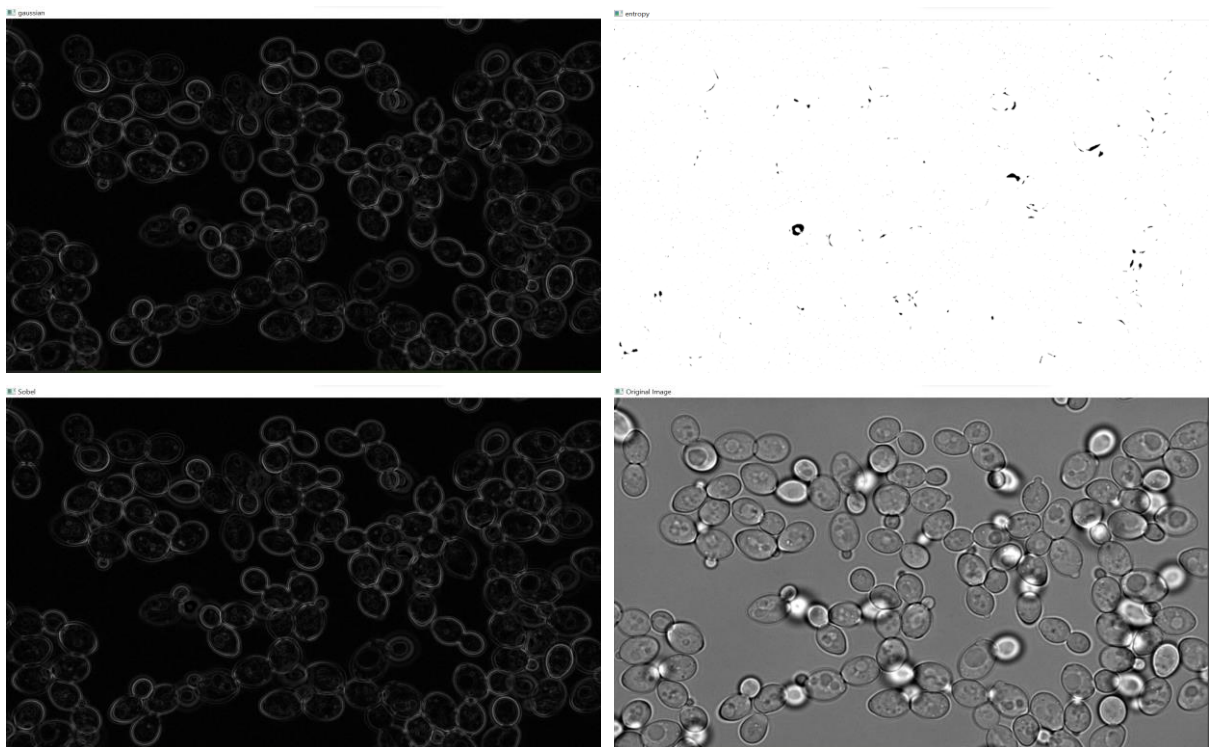

Figure 3 Different filters applied to a cell image

3.5 WHAT ARE GABOR FILTERS?

Gabor filter is a filter that contains multiple filters in one. For image processing and computer vision, Gabor filters are generally used in texture analysis, edge detection, feature extraction, etc. Gabor filters are special classes of bandpass filters, i.e., they allow a certain 'band' of frequencies and reject the others. Gabor filter takes the following parameters:

Ksize - Size of the filter returned. The kernel hast to be smaller when the features are small.

Sigma - Standard deviation of the gaussian envelope.

Theta - Orientation of the normal to the parallel stripes of a Gabor function.

Lambda - Wavelength of the sinusoidal factor.

Gamma - Spatial aspect ratio.

Psi - Phase offset.

Ktype - Type of filter coefficients. It can be CV_32F or CV_64F. It indicates the type and range of values that each pixel in the Gabor kernel can hold (float32 or float64)

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

In image processing, a Gabor filter is a linear filter used for texture analysis, which essentially means that it analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the point or region of analysis. Frequency and orientation representations of Gabor filters are claimed by many contemporary vision scientists to be similar to those of the human visual system.[1] They have been found to be particularly appropriate for texture representation and discrimination. In the spatial

domain, a 2-D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave (see Gabor transform).

Some authors claim that simple cells in the visual cortex of mammalian brains can be modeled by Gabor functions.[2][3] Thus, image analysis with Gabor filters is thought by some to be similar to perception in the human visual system.

It helps if you know what you're looking for to engineer the right filter to extract the right features. One can also use a lot of filter banks and let ML algorithms find it.
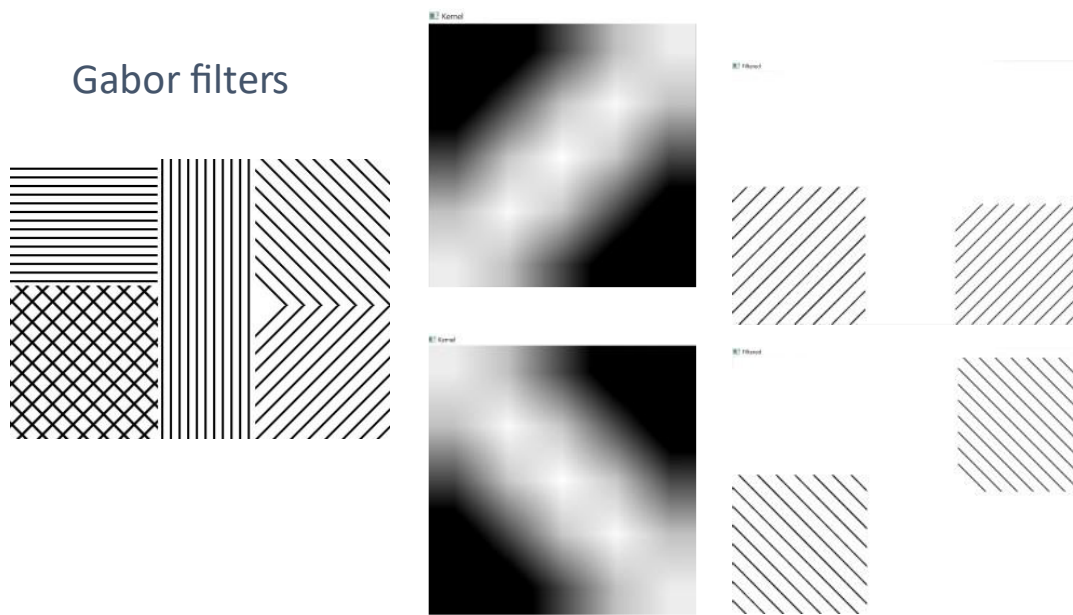

Figure 4 Gabor Filtered image: lamda = 3 , theta 45 degrees, gamma = 0.4. We filter out only lines that are 45 degrees positive/negative: we see only the lines aligned with the Gabor kernel , all the other lines are suppressed

3.6 LIST OF COMMANDS USED

```python
ksize = 5  # Use size that makes sense to the image and fetaure size.
Large may not be good.
# On the synthetic image it is clear how ksize affects imgae (try 5 and
50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 3 * np.pi / 4  # /4 shows horizontal 3/4 shows other
horizontal. Try other contributions
lamda = 1 * np.pi / 4  # 1/4 works best for angled. wavelength -
related to frequency
gamma = 0.4  # Value of 1 defines spherical. Calue close to 0 has high
aspect ratio
# Value of 1, spherical may not be ideal as it picks up features from
other regions.
phi = 0  # Phase offset. I leave it to 0. Indicates symmetry.

kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma,
phi, ktype=cv2.CV_32F)
# call Gabor filter, define Kernel size, sd, other parameters, ktype =
data type 32Float
#plt.imshow(kernel)
#plt.show()
kernel_resized = cv2.resize(kernel, (500, 500))  # Resize image
cv2.imwrite("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\
gabor.jpg",kernel_resized)
cv2.imshow('Kernel',kernel_resized)
cv2.waitKey(0)
cv2.destroyAllWindows()

img =
cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\s
ynthetic.jpg")
# img =
cv2.imread("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\B
SE.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
fimg = cv2.filter2D(img, cv2.CV_8UC3, kernel)
# this is how we apply a kernel
kernel_resized = cv2.resize(kernel, (400, 400))  # Resize image
cv2.imshow('Kernel', kernel_resized)
cv2.imshow('Original Img.', img)
cv2.imshow('Filtered', fimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. Random Forests and feature banks

4.1 RANDOM FOREST CLASSIFIERS
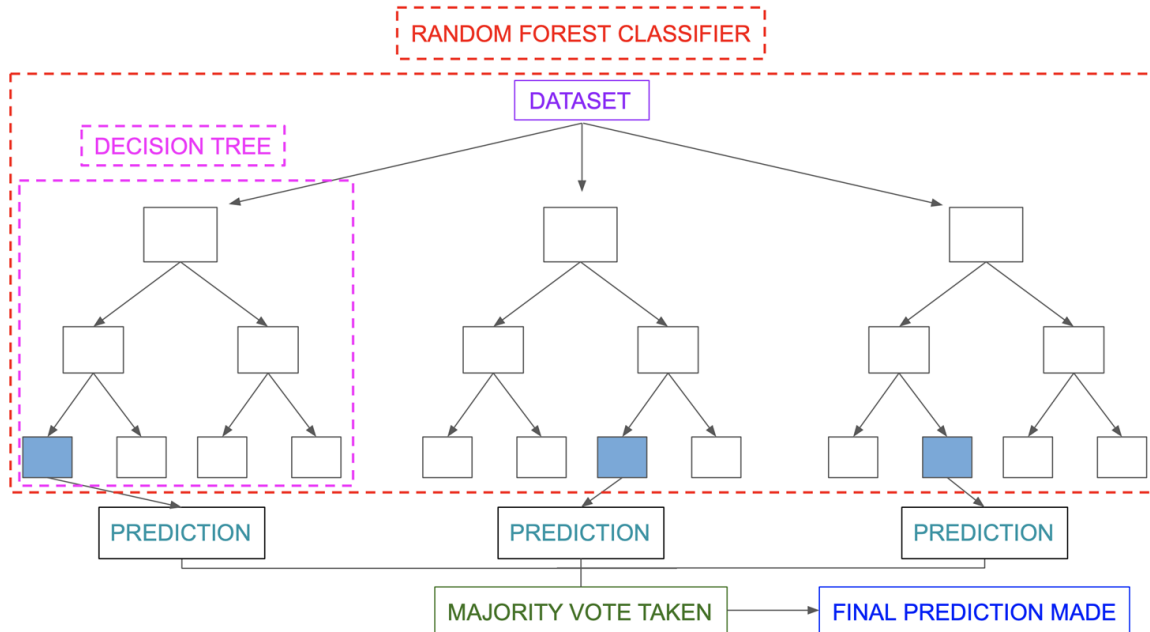
Random Forest is one of the two primary algorithms for traditional ML. It works well when we have sparse data or not a lot of data. The second one is support vector machines.
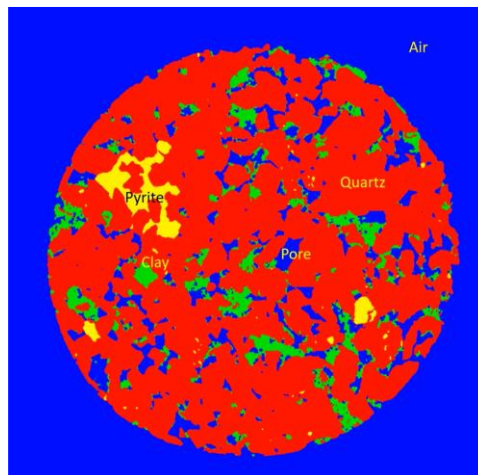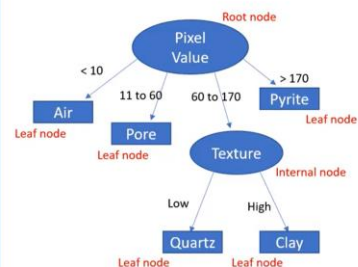
It is called forest because it is a collection of decision trees. It's part of supervised ML so it needs labelled data.

Let's study the following example from geology.

Once all possible branches in our decision are defined, we have a random forest. Nodes are metrics: root node, leaf node, internal node.

Disadvantage of decision tree: the model suffers from overfitting. It works well on training data but fails on new data leading to low accuracy. That's why we use Random Forest: a collection of decision trees. The input of a RF model is a feature vector for each pixel.

Procedure:

Taking a subset of the data 2/3 and creates a **Bootstrap** dataset: has the same size as input data contains only about 2/3 some of this input data gets duplicated. It randomly selects 2/3 of the data set.

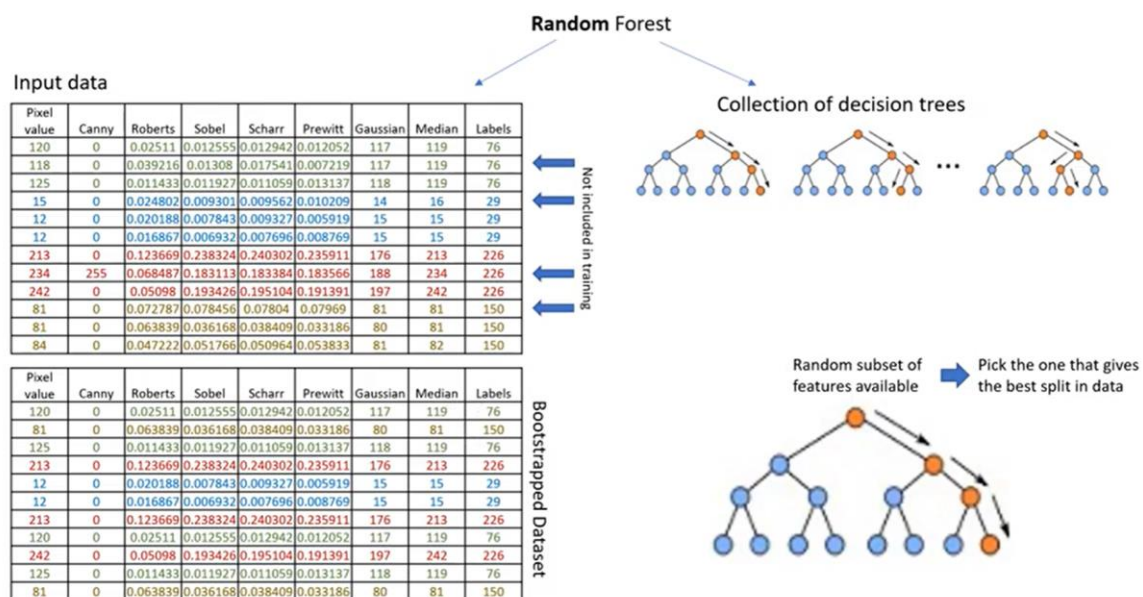Second step where randomness comes into play is **creation of decision tree**:

We randomly select a subset of features to be the primary node (root node).

In this example it will choose sqrt(9) = 3 features

Then calculated the Gini index to choose the one that gives the max info/the best split.

Verifying the accuracy of the RF : take that 1/3 (testing set) remaining and see the accuracy.

The majority wins so we attribute labels to the pixel values. We can also convert the result into a probability. We could also run a regression problem.

Random Forest

Input data

Pixel value	Canny	Roberts	Sobel	Scharr	Prewitt	Gaussian	Median	Labels
120	0	0.02511	0.012555	0.012942	0.012052	117	119	76
118	0	0.039216	0.01308	0.017541	0.007219	117	119	76
125	0	0.011433	0.011927	0.011059	0.013137	118	119	76
15	0	0.024802	0.009301	0.009562	0.010209	14	16	29
12	0	0.020188	0.007843	0.009327	0.005919	15	15	29
12	0	0.016867	0.006932	0.007696	0.008769	15	15	29
213	0	0.123669	0.238324	0.240302	0.235911	176	213	226
234	255	0.068487	0.183113	0.183384	0.183566	188	234	226
242	0	0.05098	0.193426	0.195104	0.191391	197	242	226
81	0	0.072787	0.078456	0.07804	0.07969	81	81	150
81	0	0.063839	0.036168	0.038409	0.033186	80	81	150
84	0	0.047222	0.051766	0.050964	0.053833	81	82	150

Not included in training

Collection of decision trees

Pixel value	Canny	Roberts	Sobel	Scharr	Prewitt	Gaussian	Median	Labels
120	0	0.02511	0.012555	0.012942	0.012052	117	119	76
81	0	0.063839	0.036168	0.038409	0.033186	80	81	150
125	0	0.011433	0.011927	0.011059	0.013137	118	119	76
213	0	0.123669	0.238324	0.240302	0.235911	176	213	226
12	0	0.020188	0.007843	0.009327	0.005919	15	15	29
12	0	0.016867	0.006932	0.007696	0.008769	15	15	29
213	0	0.123669	0.238324	0.240302	0.235911	176	213	226
120	0	0.02511	0.012555	0.012942	0.012052	117	119	76
242	0	0.05098	0.193426	0.195104	0.191391	197	242	226
125	0	0.011433	0.011927	0.011059	0.013137	118	119	76
81	0	0.063839	0.036168	0.038409	0.033186	80	81	150

Bootstrapped Dataset

Random subset of features available → Pick the one that gives the best split in data

4.2 HOW TO USE RANDOM FOREST IN PYTHON?

```python
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

#STEP 1: DATA READING AND UNDERSTANDING

df =
pd.read_csv("C:\\Users\\golub\\PycharmProjects\\pythonProject\\Track
module 1\\Track Module
ML\\productivity_analyzed.csv",on_bad_lines='skip')
print(df.head())


#Count productivity values to see the split between Good and Bad
sizes = df['Productivity'].value_counts(sort = 1) # sort values
print(sizes)
# The data is balanced ( almost 50/50)

#plt.pie(sizes, autopct='%1.1f%%')
#Good to know so we know the proportion of each label


#STEP 2: DROP IRRELEVANT DATA (keep independant values)
#In our example, Images_Analyzed reflects whether it is good analysis
or bad
#so should not include it. ALso, User number is just a number and has
no inflence
#on the productivity, so we can drop it.

df.drop(['Images_Analyzed'], axis=1, inplace=True) # axis =1  is column
df.drop(['User'], axis=1, inplace=True)


#STEP 3: Handle missing values, if needed
#df = df.dropna()  #Drops all rows with at least one null value.


#STEP 4: Convert non-numeric to numeric, if needed.
#Sometimes we may have non-numeric data, for example batch name, user
name, city name, etc.
#e.g. if data is in the form of YES and NO then convert to 1 and 2

df.Productivity[df.Productivity == 'Good'] = 1
df.Productivity[df.Productivity == 'Bad'] = 2
print(df.head())


#STEP 5: PREPARE THE DATA (Define dependent variable we want to
predict)

#Y is the data with dependent variable, this is the Productivity column
```

```python
Y = df["Productivity"].values  #At this point Y is an object not of
type int
#Convert Y to int
Y=Y.astype('int')

#X is data with independent variables, everything except Productivity
column
# Drop label column from X as you don't want that included as one of
the features
X = df.drop(labels = ["Productivity"], axis=1) # drop the dependent
variable
print(X.head())

#STEP 6: SPLIT THE DATA into TRAIN AND TEST data.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.4, random_state=20)
#random_state can be any integer and it is used as a seed to randomly
split dataset.
#By doing this we work with same test dataset evry time, if this is
important.
#random_state=None splits dataset randomly every time

print(X_train)

#STEP 7: Defining the model and training.

# Import the model we are using

#RandomForestRegressor is for regression type of problems.
#For classification we use RandomForestClassifier.
#Both yield similar results except for regressor the result is float
#and for classifier it is an integer.
#Let us use classifier since this is a classification problem

from sklearn.ensemble import RandomForestClassifier
#from sklearn.ensemble import RandomForestRegressor
# with Regressor result is a floating point value
# Classifier for classification and Regressor for regression

# Instantiate model with 10 decision trees
model = RandomForestClassifier(n_estimators=10, random_state=30)
# Train the model on training data
model.fit(X_train, y_train)


#STEP 8: TESTING THE MODEL BY PREDICTING ON TEST DATA
#AND CALCULATE THE ACCURACY SCORE

prediction_test = model.predict(X_test)
#print(y_test, prediction_test)

from sklearn import metrics
#Print the prediction accuracy
print ("Accuracy = ", metrics.accuracy_score(y_test, prediction_test))
```

```
#Test accuracy for various test sizes and see how it gets better with
more training data
# accuracy is displayed in percentage
# One amazing feature of Random forest is that it provides us info on
feature importances
# Get numerical feature importances
#importances = list(model.feature_importances_)

#Let us print them into a nice format.

feature_list = list(X.columns) # see what are the factors ( independent
variable)
feature_imp =
pd.Series(model.feature_importances_,index=feature_list).sort_values(as
cending=False)
print(feature_imp)
```

In summary, given a dataset with features and labels, here are the steps we have to perform:

1. Data reading and understanding

2. Drop irrelevant data (keep independent values)

3. Handle missing values, if needed

4. Convert non-numeric to numeric, if needed.

5. Prepare the data: define dependent variable we want to predict, set as integer

6. Split the data into train and test data.

7. Defining the model and training.

8. Testing the model by predicting on test data

9. Calculate the accuracy score

10. Assess feature importance

4.3 GABOR FEATURE BANKS

```python
# Generate Gabor features
num = 1  # To count numbers up in order to give Gabor features a label in the data frame
kernels = []  # Create empty list to hold all kernels that we will generate in a loop
for theta in range(2):  # Define number of thetas. Here only 2 theta values 0 and 1/4 . pi
    theta = theta / 4. * np.pi
    for sigma in (1, 3):  # Sigma with values of 1 and 3
        for lamda in np.arange(0, np.pi, np.pi / 4):  # Range of wavelengths
            for gamma in (0.05, 0.5):  # Gamma values of 0.05 and 0.5

                gabor_label = 'Gabor' + str(num)  # Label Gabor columns as Gabor1, Gabor2, etc.
                #               print(gabor_label)
                ksize = 9
                kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma, 0, ktype=cv2.CV_32F)
                # generate a kernel to convolve the image with
                kernels.append(kernel)
                # Now filter the image and add values to a new column
                fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
                filtered_img = fimg.reshape(-1)
                df[gabor_label] = filtered_img  # Labels columns as Gabor1, Gabor2, etc.
                print(gabor_label, ': theta=', theta, ': sigma=', sigma, ': lamda=', lamda, ': gamma=', gamma)
                num += 1  # Increment for gabor column label
```

The goal is to produce as an output a panda data frame with various Gabor responses from an input image that can be fed into a ML algorithm.

4.4 LIST OF COMMANDS USED

Commands	Description	Library
pd.read_csv("path.csv",on_bad_lines='skip')	Read the data frame	pandas
df.drop(['column_name'], axis=1, inplace=True)	Drop irrelevant data	pandas
df.dropna()	Drop missing values	pandas
df.column1[df.column1 == 'Good'] = 1	Setting non numeric to numeric	Pandas
print(df.head())	Have an overview of your data frame	pandas

df["Y"].values Y. astype('int')	Prepare the dependent variable	pandas
train_test_split(X, Y, test_size=0.4, random_state=20)	Split the data set into training and testing sets	sklearn.model_selection.train_test_split
RandomForestClassifier(n_estimators=10, random_state=30)	Defining the model	sklearn.ensemble.RandomForestClassifier
model.fit(X_train, y_train)	Training the model on training set	sklearn.ensemble.RandomForestClassifier
model.predict(X_test)	Predicting Y using test set	sklearn.ensemble.RandomForestClassifier
metrics.accuracy_score(y_test, prediction_test)	Accuracy of prediction	from sklearn import metrics
model.feature_importances_,index=feature_list) or with the feature names (feature_list = list(X.columns)) pd.Series(model.feature_importances_,index=feature_list).sort_values(ascending=False)	Feature importances	RandomForest
Commands	Description	Library
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)	Convert to grey level	OpenCV
pd.DataFrame()	Creating an empty data frame	pandas
img.reshape(-1)	Collapsing image into a signle column	numpy
df[,name'] = img	Filling a column in data frame	Pandas
kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda,	Create a Gabor kernel	OpenCV

gamma, 0, ktype=cv2.CV_32F)		
cv2.filter2D(img2, cv2.CV_8UC3, kernel)	Convolve the image with the kernel	OpenCV
df.to_csv("name.csv")	Write a csv file	Pandas

Commands	Description
df['Area'].plot(kind='hist', title='Area', bins=50)	Draw a histogram from data frame
pd.DataFrame(data, index = [1,2,3,4], columns = ['Area', 'Intensity', 'Orientation'])	Create a data frame
print(df.info())	Prvides an overview of the dataframe.
print(df.shape)	How many rows and columns
print(df.head())	Default prints 5 rows from the top, First default column you see are indices.
print(df.tail())	Default prints 5 rows from the bottom
df[,colname': 0'].unique())	To look at unique entrees
df = df.rename(columns = {'Unnamed: 0':'Image_set'})	Rename columns
df.describe()	Gives statistical summary of each column.
df1 = df.drop("Manual2", axis=1)	Creating a new dataframe df1. Axis=1 means referring to column.
df['Date'] = pd.to_datetime("2022-11-13")	Set date as date type
df.to_csv(file_name.csv')	Write a data frame to csv

df.sort_values('Manual', ascending=True)	Sort values from lowest to highest
df[20:30]	Select specific rows
df.loc[20:30, ['colname', 'Auto_th_2']])	Get specific columns from specific rows
max(df[,colname'])	Gives maximum value in a given column
df['colname'] > 100.)	Select values, prints True or False
df.groupby(by=[,colname'])	Group-by's can be used to build groups of rows based off a specific feature
df.corr()	Correlation between all columns
df.isnull())	Shows whether a cell is null or not
df.isnull().sum()	Shows number of nulls in each column.
df.dropna()	Drops all rows with at least one null value.
print(df['colname'].describe())	Gives statistics e.g. mean
df['colname'].fillna(100, inplace=True)	Fill na with means for example (imputation)
df['colnamel'].plot(kind="box", figsize=(8,6)) df.boxplot(column='name_col', by='index')	Boxplot, Shows max and min values, outliers
df.plot(kind='scatter', x='x_name', y='y_name', title='main title')	Scatter plot to see relationship between columns
df["colname"].apply(function)	Apply function to the whole column

5. Image Segmentation using traditional machine learning

In this chapter, we are going to perform image segmentation using Random Forest Classifier.

We're dealing with labeled images; every pixel is assigned a specific label. Then we randomly divide the data into two subsets: training and testing. We'll then put feature extractors/ filters on it to create a feature vector. Find out which filters are relevant. We'll then feed that into our ML algorithm (RF). It gives us a trained model. To check if the model is fine, we'll validate it with the testing model. We'll then save the model as pickle file (for long-term storage). We can then use it for segmenting future images. Here's the plan for the procedure:

1)   Feature extraction

2)   Create ML model and validate it

3)   Relevant features?

4)   Save the result

5)   Segmentation

5.1 FEATURE EXTRACTION

Let's use ML instead of engineering features manually. We create a feature vector using previously defined Gabor filter bank and some additional filters. We create a data frame with features, original pixels and labels.

```
...  print(df.head())
   Original Image  Gabor1  Gabor2  ...  Median s3  Variance s3  Labels
0               0       0       0  ...          0            0      29
1               0       0       0  ...          0            0      29
2               0       0       0  ...          0            0      29
3               0       0       0  ...          0            0      29
4               0       0       0  ...          0            0      29

[5 rows x 39 columns]
```

5.2 TRAINING THE RANDOM FOREST CLASSIFIER

After splitting the datasets, create a model with RF Classifier, fit it on the training test and predict and compare with the test set.

```python
Y = df['Labels'].values
X = df.drop(labels =['Labels'], axis =1)
# Split data into test and train
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.4, random_state = 20)

# import ML algorithm and train the model
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators =10, random_state=30)
model.fit(X_train, Y_train)
# Predict asd check accuracy
from sklearn import metrics
prediction_test = model.predict(X_test)
print("Accuracy =", metrics.accuracy_score(Y_test,prediction_test))
```

5.3 FEATURE RANKING

```python
# Which feature worked the best ?
importances = list(model.feature_importances_)
features_list = list(X.columns)
feature_importance =pd.Series(model.feature_importances_,index = features_list).sort_values(ascending = False)

print(feature_importance)
# Only take top 5 or top 10
```

Once when have the list of feature importance, we can keep the top 5-10 most important once and drop the rest so the model only has the features that contribute the most to the prediction.

```
>>> print(feature_importance)
Median s3          0.185643
Gaussian s7        0.172621
Gabor5             0.163603
Gabor4             0.157023
Gaussian s3        0.101415
Gabor6             0.090871
Original Image     0.025626
Gabor24            0.018449
Gabor8             0.017576
Gabor7             0.013069
Gabor21            0.009203
>>>
```

5.4 SAVING THE MODEL

Store the model and use it for future purposes.

```python
import pickle
filename = 'stone_model'
pickle.dump(model, open(filename, 'wb'))
# copy to a location, w-write, b - binary


load_model = pickle.load(open(filename, 'rb'))
result = load_model.predict(X)
segmented = result.reshape((img.shape)) # original shape
```

5.5 SEGMENTING MULTIPLE IMAGES USING THE MODEL

We have a set of images available in the « path », that are « training images » we would like to segment. We apply the feature extraction function discussed earlier. We then call the saved model and apply it to the feature vectors of each of the images and store the segmented images in the « predicted images » folder.

```python
def feature_extraction(img):
    df = pd.DataFrame()
    df['Original Image'] = img.reshape(-1)
    # Generate Gabor filters
    num = 1  # To count numbers up in order to give Gabo
    kernels = []  # Create empty list to hold all kernel
    for theta in range(2):...
    edges = cv2.Canny(img,100,200)
    df['Canny edge'] = edges.reshape(-1)
    from scipy import ndimage as nd
    gaussian_3 = nd.gaussian_filter(img, sigma=3)
    df['Gaussian s3'] = gaussian_3.reshape(-1)
    gaussian_7 = nd.gaussian_filter(img, sigma=7)
    df['Gaussian s7'] = gaussian_3.reshape(-1)
    median_img = nd.median_filter(img, size = 3)
    df['Median s3']= median_img.reshape(-1)
    return df
```

```python
filename = 'stone_model'
load_model = pickle.load(open(filename,'rb'))
path = "C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\Stones\\Training images\\*.tif"
for file in glob.glob(path):
    img = cv2.imread(file)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    X = feature_extraction(img)
    result = load_model.predict(X)
    segmented = result.reshape(img.shape)
    name = file.split("e_")
    cm = plt.get_cmap('jet') # Apply the colormap like a function to any array:
    colored = cm(img)
    tifffile.imwrite("C:\\Users\\golub\\PycharmProjects\\pythonProject\\images\\Stones\\Predicted images\\" + name[1], colored)
```

We finally have a model that automate the process of segmenting images.
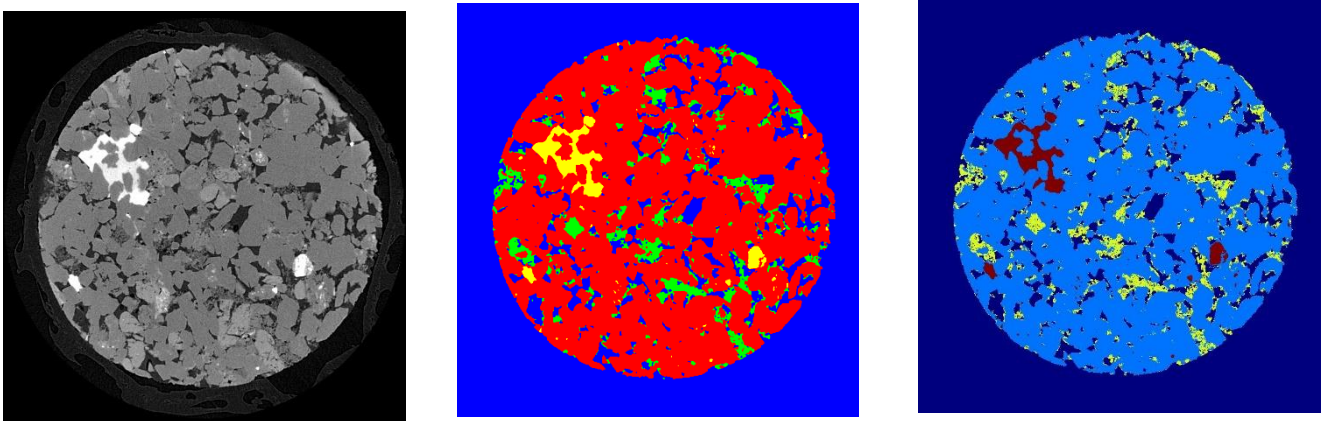We can now see clear patterns and classify them.

Figure 5 Training image of the stone, original image and predicted segmentation.

References

Digital Sreeni channel https://www.youtube.com/@DigitalSreeni/featured

Sreeni's Github https://github.com/bnsreenu/python_for_microscopists

Gini index https://www.analyticssteps.com/blogs/what-gini-index-and-information-gain-decision-trees

Image denoising using DL https://towardsai.net/p/deep-learning/image-de-noising-using-deep-learning

Non-local means denoising https://scikit-image.org/docs/stable/auto_examples/filters/plot_nonlocal_means.html

Nd Filters in Scipy
https://docs.scipy.org/doc/scipy/reference/ndimage.html?highlight=filter#

Colormaps in Matplotlib
https://matplotlib.org/stable/tutorials/colors/colormaps.html